

---

# **cepton\_sdk Documentation**

**Cepton Technologies**

**Dec 07, 2018**



# CONTENTS

<b>1</b>	<b>Overview</b>	<b>1</b>
<b>2</b>	<b>Errors</b>	<b>3</b>
<b>3</b>	<b>Setup</b>	<b>7</b>
<b>4</b>	<b>Sensors</b>	<b>11</b>
<b>5</b>	<b>Points</b>	<b>13</b>
<b>6</b>	<b>Networking</b>	<b>15</b>
<b>7</b>	<b>Capture Replay</b>	<b>17</b>
<b>8</b>	<b>API</b>	<b>19</b>
<b>9</b>	<b>Utilities</b>	<b>21</b>
<b>10</b>	<b>Samples</b>	<b>25</b>
	<b>Index</b>	<b>35</b>



## OVERVIEW

The SDK parses sensor packets and computes calibrated measurements from raw sensor data. For prototyping purposes, it can optionally handle networking, frame accumulation, and capture replay.

For prototyping, it is recommended to start with the high level methods in *API* and *Utilities*.

### 1.1 Timestamps

All `int64` timestamps are microseconds since the Unix epoch (UTC). Point timestamps are based on one of the following sources (the first valid source is used):

1. GPS (NMEA + PPS)
2. PTP
3. Host PC

All `float` and `int64` time differences are seconds (measurement period, replay time, frame length, etc.).

### 1.2 Multiple Returns

To enable multiple returns, pass the `CEPTON_SDK_CONTROL_ENABLE_MULTIPLE_RETURNS` flag during initialization.

The returns are as follows:

1. Strongest signal.
2. Furthest signal, if it is not the strongest. Otherwise, the second strongest signal.

### 1.3 Concurrency

- If networking is enabled, the sdk creates 2 threads for networking. All callbacks occur in these networking threads.
- If networking is disabled, the sdk does not create any threads. All callbacks occur in the main thread.
- All sdk getter functions are thread safe, and can be called from callbacks. Other sdk functions are not guaranteed to be thread safe, and can cause deadlock if called from callbacks.

## 1.4 Minimal SDK

If desired, the following SDK features can be disabled in the SDK and performed manually by the user:

- Networking: see *Network*.
- Frame Accumulation: see *Frame*.

## ERRORS

## 2.1 Types

**class** **SensorError** : **public** **runtime\_error**

Error returned by most functions.

Implicitly convertible from/to *SensorErrorCode*. Getter functions do not return an error, because they cannot fail.

### Public Functions

**SensorError** (*SensorErrorCode* *code\_*, **const** **char** \***const** *msg\_*)

**SensorError** (*SensorErrorCode* *code*)

**SensorError** ()

**operator bool** () **const**

Returns *false* if code is *CEPTON\_SUCCESS*, true otherwise.

**operator SensorErrorCode** () **const**

**const** **char** \***name** () **const**

**bool is\_error** () **const**

**bool is\_fault** () **const**

### Public Members

*SensorErrorCode* **code**

**std::string** **msg**

**enum** **\_CeptonSensorErrorCode**

*Values:*

**CEPTON\_SUCCESS** = 0

**CEPTON\_ERROR\_GENERIC** = -1

**CEPTON\_ERROR\_OUT\_OF\_MEMORY** = -2

**CEPTON\_ERROR\_SENSOR\_NOT\_FOUND** = -4

**CEPTON\_ERROR\_SDK\_VERSION\_MISMATCH** = -5

**CEPTON\_ERROR\_COMMUNICATION** = -6

Networking error.

**CEPTON\_ERROR\_TOO\_MANY\_CALLBACKS** = -7

**CEPTON\_ERROR\_INVALID\_ARGUMENTS** = -8

Invalid value or uninitialized struct.

**CEPTON\_ERROR\_ALREADY\_INITIALIZED** = -9

**CEPTON\_ERROR\_NOT\_INITIALIZED** = -10

**CEPTON\_ERROR\_INVALID\_FILE\_TYPE** = -11

**CEPTON\_ERROR\_FILE\_IO** = -12

**CEPTON\_ERROR\_CORRUPT\_FILE** = -13

**CEPTON\_ERROR\_NOT\_OPEN** = -14

**CEPTON\_ERROR\_EOF** = -15

**CEPTON\_FAULT\_INTERNAL** = -1000

Internal parameter out of range.

**CEPTON\_FAULT\_EXTREME\_TEMPERATURE** = -1001

Reading exceed spec.

**CEPTON\_FAULT\_EXTREME\_HUMIDITY** = -1002

Reading exceeds spec.

**CEPTON\_FAULT\_EXTREME\_ACCELERATION** = -1003

**CEPTON\_FAULT\_ABNORMAL\_FOV** = -1004

**CEPTON\_FAULT\_ABNORMAL\_FRAME\_RATE** = -1005

**CEPTON\_FAULT\_MOTOR\_MALFUNCTION** = -1006

**CEPTON\_FAULT\_LASER\_MALFUNCTION** = -1007

**CEPTON\_FAULT\_DETECTOR\_MALFUNCTION** = -1008

**typedef** CeptonSensorErrorCode cepton\_sdk::SensorErrorCode

**typedef** int32\_t CeptonSensorErrorCode

## 2.2 Methods

**const** char\* cepton\_sdk::get\_error\_code\_name (*SensorErrorCode* error\_code)

Returns string name of error code.

Returns empty string if error code is invalid.

**bool** cepton\_sdk::is\_error\_code (*SensorErrorCode* error\_code)

Returns true if error name is of the form CEPTON\_ERROR\_\*, false otherwise.

**bool** cepton\_sdk::is\_fault\_code (*SensorErrorCode* error\_code)

Returns true if error name is of the form CEPTON\_FAULT\_\*, false otherwise.



*SensorError* cepton\_sdk::get\_error()

Returns and clears the last sdk error.

Called automatically by all C++ methods, so only useful when calling C methods directly.



## 3.1 Types

**enum \_CeptonSDKControl**

SDK control flags.

*Values:*

**CEPTON\_SDK\_CONTROL\_DISABLE\_NETWORK** = 1 << 1

Disable networking operations.

Useful for running multiple instances of sdk in different processes. Must pass packets manually to `cepton_sdk::mock_network_receive`.

**CEPTON\_SDK\_CONTROL\_DISABLE\_IMAGE\_CLIP** = 1 << 2

Disable marking image clipped points as invalid.

Does not affect number of points returned.

**CEPTON\_SDK\_CONTROL\_DISABLE\_DISTANCE\_CLIP** = 1 << 3

Disable marking distance clipped points as invalid.

Does not affect number of points returned.

**CEPTON\_SDK\_CONTROL\_ENABLE\_MULTIPLE\_RETURNS** = 1 << 4

Enable multiple returns.

When set, `cepton_sdk::SensorInformation::return_count` will indicate the number of returns per laser. Can only be set at sdk initialization.

**CEPTON\_SDK\_CONTROL\_ENABLE\_STRAY\_FILTER** = 1 << 5

Enable marking stray points as invalid (measurement noise).

Uses `cepton_sdk::util::StrayFilter` to mark points invalid.

Does not affect number of points returned.

**CEPTON\_SDK\_CONTROL\_HOST\_TIMESTAMPS** = 1 << 6

Always use packet timestamps (disable GPS/PTP timestamps).

**typedef** CeptonSDKControl `cepton_sdk::Control`

**typedef** uint32\_t **CeptonSDKControl**

**enum \_CeptonSDKFrameMode**

Controls frequency of points being reported.

*Values:*

**CEPTON\_SDK\_FRAME\_STREAMING** = 0

Report points by packet.

**CEPTON\_SDK\_FRAME\_TIMED** = 1

Report points at fixed time intervals.

Interval controlled by *CeptonSDKFrameOptions::length*.

**CEPTON\_SDK\_FRAME\_COVER** = 2

Report points when the field of view is covered once.

- For HR80 series, detects half scan cycle (left-to-right or right-to-left).

**CEPTON\_SDK\_FRAME\_CYCLE** = 3

Report points when the scan pattern goes through a full cycle.

Typically 2x longer frame than COVER mode.

- For HR80 series, detects full scan cycle (left-to-right-to-left).
- For VISTA series, internally uses TIMED mode.

**CEPTON\_SDK\_FRAME\_MODE\_MAX** = 3

**typedef** CeptonSDKFrameMode cepton\_sdk::FrameMode

**typedef** uint32\_t CeptonSDKFrameMode

**struct** CeptonSDKFrameOptions

### Public Members

size\_t **signature**

Internal use only.

*CeptonSDKFrameMode* **mode**

Default: CEPTON\_SDK\_FRAME\_STREAMING.

float **length**

Frame length [seconds].

Default: 0.05. Only used if mode=CEPTON\_SDK\_FRAME\_TIMED.

**typedef** CeptonSDKFrameOptions cepton\_sdk::FrameOptions

*FrameOptions* cepton\_sdk::create\_frame\_options()

Create default frame options.

**struct** CeptonSDKOptions

SDK initialization options.

### Public Members

size\_t **signature**

Internal use only.

*CeptonSDKControl* **control\_flags**

Default: 0.

**struct** *CeptonSDKFrameOptions* **frame**

`uint16_t port`  
Default: 8808.

**typedef** CeptonSDKOptions cepton\_sdk::Options

*Options* cepton\_sdk::create\_options()  
Create default options.

**typedef** void (\*cepton\_sdk::FpSensorErrorCallback) (*SensorHandle* handle, *SensorErrorCode* error\_code, **const** char \*error\_msg, **const** void \*error\_data, size\_t error\_data\_size, void \*user\_data)

Callback for receiving sdk and sensor errors.

Currently, error\_data is not used.

## 3.2 Methods

*SensorError* cepton\_sdk::initialize (int version, **const** *Options* &options = create\_options(), **const** *FpSensorErrorCallback* &cb = nullptr, void \***const** user\_data = nullptr)

Initializes settings and networking.

Must be called before any other sdk function listed below.

*SensorError* cepton\_sdk::deinitialize()  
Resets everything and deallocates memory.

Called automatically on program exit.

*SensorError* cepton\_sdk::set\_control\_flags (*Control* mask, *Control* flags)

*Control* cepton\_sdk::get\_control\_flags()

bool cepton\_sdk::has\_control\_flag (*Control* flag)

uint16\_t cepton\_sdk::get\_port()

*SensorError* cepton\_sdk::set\_port (uint16\_t port)  
Sets network listen port.

Default: 8808.

*SensorError* cepton\_sdk::set\_frame\_options (**const** CeptonSDKFrameOptions &options)

*FrameMode* cepton\_sdk::get\_frame\_mode()

float cepton\_sdk::get\_frame\_length()



## SENSORS

## 4.1 Types

```
typedef CeptonSensorHandle cepton_sdk::SensorHandle  
    Sensor identifier.
```

```
enum _CeptonSensorModel
```

*Values:*

```
    HR80T = 1
```

```
    HR80M = 2
```

```
    HR80W = 3
```

```
    SORA_200 = 4
```

```
    VISTA_860 = 5
```

```
    HR80T_R2 = 6
```

```
    VISTA_860_GEN2 = 7
```

```
    FUSION_790 = 8
```

```
    CEPTON_SENSOR_MODEL_MAX = 8
```

```
typedef CeptonSensorModel cepton_sdk::SensorModel
```

```
typedef CeptonSensorInformation cepton_sdk::SensorInformation
```

```
struct CeptonSensorInformation
```

### Public Members

```
CeptonSensorHandle handle
```

```
uint64_t serial_number
```

```
char CeptonSensorInformation::model_name[28]
```

```
CeptonSensorModel model
```

```
char CeptonSensorInformation::firmware_version[32]
```

```
float last_reported_temperature  
    [celsius]
```

```
float last_reported_humidity  
    [%]
```

```

float last_reported_age
    [hours]

float measurement_period
    Time between measurements [seconds].

int64_t ptp_ts
    [microseconds]

uint8_t gps_ts_year
    0-99 (2017 -> 17)

uint8_t gps_ts_month
    1-12

uint8_t gps_ts_day
    1-31

uint8_t gps_ts_hour
    0-23

uint8_t gps_ts_min
    0-59

uint8_t gps_ts_sec
    0-59

uint8_t return_count

uint8_t segment_count
    Number of image segments.

uint32_t flags

```

## 4.2 Methods

```

std::size_t cepton_sdk::get_n_sensors()
    Get number of sensors attached. Use to check for new sensors. Sensors are not deleted until deinitialization.

SensorError cepton_sdk::get_sensor_handle_by_serial_number(uint64_t serial_number,
    SensorHandle &handle)
    Returns error if sensor not found.

SensorError cepton_sdk::get_sensor_information_by_index(std::size_t idx, SensorInformation &info)
    Valid indices are in range [0, n_sensors). Returns error if index invalid.

SensorError cepton_sdk::get_sensor_information(SensorHandle handle, SensorInformation &info)
    Returns error if sensor not found.

```



## 5.1 Types

**typedef** CeptonSensorImagePoint cepton\_sdk::SensorImagePoint

**struct** CeptonSensorImagePoint

Point in image coordinates (focal length = 1).

To convert to 3d point, refer to `ception_sdk_util.hpp`.

Subclassed by *ception\_sdk::util::SensorPoint*

### Public Members

int64\_t **timestamp**

Unix time [microseconds].

float **image\_x**

x image coordinate.

float **distance**

Distance [meters].

float **image\_z**

z image coordinate.

float **intensity**

Diffuse reflectance.

CeptonSensorReturnType **return\_type**

uint8\_t **flags**

bit flags

1. **valid**: If `false`, then the distance and intensity are invalid.
2. **saturated**: If `true`, then the intensity is invalid. Also, the distance is valid, but inaccurate.

## 5.2 Methods

```
typedef void (* FpCeptonSensorImageDataCallback) (CeptonSensorHandle handle,  
size_t n_points, const struct Cep-  
tonSensorImagePoint *c_points,  
void *user_data)
```

*SensorError* cepton\_sdk::**listen\_image\_frames** (FpSensorImageDataCallback *cb*, void \***const** *user\_data*)

Sets image frames callback.

Returns points at frequency specified by *cepton\_sdk::FrameOptions::mode*. Each frame contains all possible points (use *cepton\_sdk::SensorImagePoint::valid* to filter points). Points are ordered by measurement, segment, and return:

```
measurement_count = n_points / (segment_count * return_count)
idx = ((i_measurement) * segment_count + i_segment) * return_count + i_return
```

Returns error if callback already registered.

*SensorError* cepton\_sdk::**unlisten\_image\_frames** ()

## NETWORKING

## 6.1 Types

```
typedef void (*cepton_sdk::FpNetworkReceiveCallback) (SensorHandle handle, int64_t  
                                                    timestamp, uint8_t const  
                                                    *buffer, size_t buffer_size, void  
                                                    *user_data)
```

Callback for receiving network packets.

### Parameters

- `handle`: Unique sensor identifier (e.g. IP address). Returns error if callback already set.

## 6.2 Methods

```
SensorError cepton_sdk::listen_network_packets (FpNetworkReceiveCallback cb, void  
                                                *const user_data)
```

Sets network packets callback.

Only 1 callback can be registered.

```
SensorError cepton_sdk::unlisten_network_packets ()
```

```
SensorError cepton_sdk::clear ()
```

Clears sensors.

Use when loading/unloading capture file.



## CAPTURE REPLAY

PCAP capture file replay. Source code can be found in the *source* folder. Functions are not thread safe, and should only be called from the main thread.

`bool cepton_sdk::capture_replay::is_open()`

*SensorError* `ception_sdk::capture_replay::open(const std::string &path)`  
Opens capture file.

Must be called before any other replay functions listed below.

*SensorError* `ception_sdk::capture_replay::close()`

`const char *ception_sdk::capture_replay::get_filename()`

`uint64_t cepton_sdk::capture_replay::get_start_time()`  
Returns capture start timestamp (unix time [microseconds]).

`float cepton_sdk::capture_replay::get_position()`  
Returns capture file position [seconds].

`float cepton_sdk::capture_replay::get_length()`  
Returns capture file length [seconds].

`bool cepton_sdk::capture_replay::is_end()`  
Returns true if at end of capture file.

This is only relevant when using `resume_blocking` methods.

*SensorError* `ception_sdk::capture_replay::seek(float position)`  
Seek to capture file position [seconds].

Position must be in range [0.0, capture length). Returns error if position is invalid.

*SensorError* `ception_sdk::capture_replay::set_enable_loop(bool value)`  
If enabled, replay will automatically rewind at end.

`bool cepton_sdk::capture_replay::get_enable_loop()`

*SensorError* `ception_sdk::capture_replay::set_speed(float speed)`  
Replay speed multiplier for asynchronous replay.

`float cepton_sdk::capture_replay::get_speed()`

*SensorError* `ception_sdk::capture_replay::resume_blocking_once()`  
Replay next packet in current thread without sleeping.

Pauses replay thread if it is running.

*SensorError* `ception_sdk::capture_replay::resume_blocking(float duration)`  
Replay multiple packets synchronously.

No sleep between packets. Resume duration must be non-negative. Pauses replay thread if it is running.

`bool cepton_sdk::capture_replay::is_running()`

Returns true if replay thread is running.

*SensorError* `cepton_sdk::capture_replay::resume()`

Packets are replayed in realtime. Replay thread sleeps in between packets.

*SensorError* `cepton_sdk::capture_replay::pause()`

Pauses asynchronous replay thread.

High level SDK api for prototyping (*cepton\_sdk\_api.hpp*). Methods are agnostic to live/replay mode.

`bool cepton_sdk::api::is_live()`  
Returns true if capture replay is not open.

`bool cepton_sdk::api::is_end()`

`int64_t cepton_sdk::api::get_time()`  
Returns capture replay time or live time.

*SensorError* `cepton_sdk::api::wait` (float *t\_length* = -1.0f)  
Sleeps or resumes capture replay for duration.  
If *t\_length* < 0, then waits forever.

## 8.1 Errors

`const SensorError &cepton_sdk::api::check_error` (const *SensorError* &*error*, const std::string &*msg* = "")

Handles error.

If error, raises exception. Otherwise, prints error.

`const SensorError &cepton_sdk::api::log_error` (const *SensorError* &*error*, const std::string &*msg* = "")

Prints error.

`void cepton_sdk::api::default_on_error` (*SensorHandle* *h*, *SensorErrorCode* *error\_code*, const char \*const *error\_msg*, const void \*const *error\_data*, std::size\_t *error\_data\_size*, void \*const *instance*)

Basic SDK error callback.

Calls `cepton_sdk::api::check_error_code`.

## 8.2 Setup

*SensorError* `cepton_sdk::api::initialize` (*Options* *options* = *create\_options*(), const std::string &*capture\_path* = "")

Initialize SDK and optionally starts capture replay.

`class SensorErrorCallback : public cepton_sdk::util::Callback<SensorHandle, const SensorError>`  
Callback for sensor errors.

### Public Static Functions

```
static void global_on_callback(SensorHandle handle, SensorErrorCode error_code, const
                             char *error_msg, const void *const error_data, size_t er-
                             ror_data_size, void *const instance)
```

```
class SensorImageFrameCallback : public cepton_sdk::util::Callback<SensorHandle, std::size_t, const SensorImagePo
```

Callback for image frames.

Must call `initialize` before use.

### Public Functions

```
~SensorImageFrameCallback()
```

```
SensorError initialize()
```

```
SensorError deinitialize()
```

```
class NetworkPacketCallback : public cepton_sdk::util::Callback<SensorHandle, int64_t, uint8_t const *, std::size_t>
```

Callback for network packets.

Must call `initialize` before use.

### Public Functions

```
~NetworkPacketCallback()
```

```
SensorError initialize()
```

```
SensorError deinitialize()
```

## 8.3 Sensors

```
bool cepton_sdk::api::has_sensor_by_serial_number(uint64_t serial_number)
```

```
SensorError cepton_sdk::api::get_sensor_information_by_serial_number(uint64_t se-
                                                                    rial_number,
                                                                    SensorIn-
                                                                    formation
                                                                    &info)
```

Returns error if sensor not found.

```
std::vector<uint64_t> cepton_sdk::api::get_sensor_serial_numbers()
```

Returns serial numbers for all sensors.



## UTILITIES

Utility functions and classes for prototyping (*cepton\_sdk\_util.hpp*).

## 9.1 Common

```
int64_t cepton_sdk::util::get_timestamp_usec()
```

Returns current unix timestamp [microseconds].

This is the timestamp format used by all sdk functions.

## 9.2 Points

```
void cepton_sdk::util::convert_image_point_to_point (float image_x, float image_z, float
                                                    distance, float &x, float &y, float
                                                    &z)
```

Convert image point to 3d point.

```
struct SensorPoint : public CeptonSensorImagePoint
```

3d point class.

### Public Members

float **x**

float **y**

float **z**

```
void cepton_sdk::util::convert_sensor_image_point_to_point (const SensorImage-
                                                            Point &image_point,
```

Convenience method to convert cepton\_sdk::SensorImagePoint to  
*SensorPoint* &point)

## 9.3 Transforms

```
class CompiledTransform
```

3d translation and rotation.

For more functionality, use Eigen's Geometry module.

## Public Functions

void **apply** (float &x, float &y, float &z)  
Apply transformation to 3d position.

## Public Static Functions

**static** *CompiledTransform* **create** (**const** float \***const** translation, **const** float \***const** rotation)  
Create from translation and rotation.

### Parameters

- translation: Cartesian (x, y, z)
- rotation: Quaternion (x, y, z, w)

## 9.4 Callbacks

**template** <typename... TArgs>

**class** **Callback**

Expands SDK callback functionality.

Allows for multiple callbacks to be registered. Allows for registering lambdas and member functions. See `samples/basic.cpp`.

## Public Functions

void **clear** ()  
Clear all listeners.

*SensorError* **listen** (**const** std::function<void> TArgs...  
> &func, uint64\_t \***const** id = nullptr) Register std::function.

**template** <typename TClass>  
*SensorError* **listen** (TClass \***const** instance, MemberFunction<TClass, TArgs...> func, uint64\_t  
\***const** id = nullptr)  
Register instance member function.

void **emit** (TArgs... args) **const**  
Emit callback.

void **operator** () (TArgs... args) **const**  
Emit callback.

## Public Static Functions

**static** void **global\_on\_callback** (TArgs... args, void \***const** instance)  
Used for registering as c callback.

## 9.5 Frames

### class **FrameDetector**

Detects frames in streaming sensor data.

#### Public Functions

**FrameDetector** (*const SensorInformation* &sensor\_info)

*const FrameOptions* &get\_options () **const**

*SensorError* set\_options (*const FrameOptions* &options)

void **reset** ()

Completely resets detector.

Only use if also clearing points accumulator.

bool **add\_point** (*const SensorImagePoint* &point)

Returns true if frame found.

Automatically resets after frame is found.

#### Public Members

bool **frame\_found**

int **frame\_idx**

float **frame\_x**

Number of points in current frame.

### class **FrameAccumulator**

Accumulates image points, and emits frames to callback.

See `samples/frame.cpp`.

#### Public Functions

**FrameAccumulator** (*const SensorInformation* &sensor\_info)

*FrameOptions* get\_options () **const**

*SensorError* set\_options (*const FrameOptions* &options)

void **clear** ()

void **add\_points** (int n\_points, *const SensorImagePoint* \*const image\_points)

#### Public Members

*Callback*<int, *const SensorImagePoint* \*> **callback**



## SAMPLES

## 10.1 Basic

Listing 1: samples/basic.cpp

```
1  /**
2   * Sample code for general sdk usage.
3   */
4  #include <iostream>
5  #include <string>
6  #include <vector>
7
8  #include <cepton_sdk_api.hpp>
9
10 class FramesListener {
11 public:
12     void on_image_frame(cepton_sdk::SensorHandle handle, std::size_t n_points,
13                        const cepton_sdk::SensorImagePoint *c_image_points) {
14         // Get sensor info
15         cepton_sdk::SensorInformation sensor_info;
16         cepton_sdk::api::check_error(
17             cepton_sdk::get_sensor_information(handle, sensor_info));
18
19         // Print info
20         if (i_frame < 5) {
21             std::printf("Received %i points from sensor %i\n", (int)n_points,
22                        (int)sensor_info.serial_number);
23         }
24         ++i_frame;
25     }
26
27 private:
28     std::size_t i_frame = 0;
29 };
30
31 int main(int argc, char **argv) {
32     std::string capture_path;
33     if (argc >= 2) capture_path = argv[1];
34
35     // Initialize
36     auto options = cepton_sdk::create_options();
37     cepton_sdk::api::check_error(
38         cepton_sdk::api::initialize(options, capture_path));
39 }
```

(continues on next page)

(continued from previous page)

```

39
40 // Get sensor
41 std::printf("Waiting for sensor to connect...\n");
42 while (cepton_sdk::get_n_sensors() == 0)
43     cepton_sdk::api::check_error(cepton_sdk::api::wait(0.1f));
44 cepton_sdk::SensorInformation sensor_info;
45 cepton_sdk::api::check_error(
46     cepton_sdk::get_sensor_information_by_index(0, sensor_info));
47 std::printf("Sensor: %d\n", (int)sensor_info.serial_number);
48
49 // Listen for frames
50 std::printf("Listening for frames...\n");
51 cepton_sdk::api::SensorImageFrameCallback callback;
52 cepton_sdk::api::check_error(callback.initialize());
53 FramesListener frames_listener;
54 callback.listen(&frames_listener, &FramesListener::on_image_frame);
55
56 // Run
57 cepton_sdk::api::check_error(cepton_sdk::api::wait());
58
59 // Deinitialize (optional)
60 cepton_sdk::api::check_error(cepton_sdk::deinitialize());
61 }

```

## 10.2 C Basic

Listing 2: samples/c\_basic.c

```

1  /**
2   * Sample code for general C sdk usage.
3   */
4  #include <stdio.h>
5  #include <time.h>
6
7  #include <cepton_sdk.h>
8
9  int frames_got = 0;
10 time_t first_frame_time = 0;
11 time_t current_frame_time = 0;
12
13 void image_frame_callback(CeptonSensorHandle handle, size_t n_points,
14                          const struct CeptonSensorImagePoint *c_points,
15                          void *user_data) {
16     time_t t = time(NULL);
17
18     if (frames_got == 0) first_frame_time = t;
19     frames_got++;
20     if (frames_got < 50)
21         printf("Frame: %4d Time: %lld\n", frames_got, (long long)t);
22     else
23         printf("Frame: %4d Time: %lld Frame rate: %.1fHz\n", frames_got,
24             (long long)t, frames_got * 1.0 / (t - first_frame_time));
25 }
26

```

(continues on next page)

(continued from previous page)

```

27 int main() {
28     printf("Start\n");
29     fflush(stdout);
30     struct CeptonSDKOptions options = cepton_sdk_create_options();
31     options.frame.mode = CEPTON_SDK_FRAME_TIMED;
32     options.frame.length = 0.1f;
33     CeptonSensorErrorCode ret;
34     ret = cepton_sdk_initialize(CEPTON_SDK_VERSION, &options, NULL, NULL);
35     if (ret != CEPTON_SUCCESS) printf("%s\n", cepton_get_error_code_name(ret));
36     ret = cepton_sdk_listen_image_frames(image_frame_callback, NULL);
37     if (ret != CEPTON_SUCCESS) printf("%s\n", cepton_get_error_code_name(ret));
38
39     while (frames_got < 100)
40         ; // Just spin loop for lack of cross platform sleep in C
41
42     cepton_sdk_deinitialize();
43
44     return 0;
45 }

```

## 10.3 Callback

Listing 3: samples/callback.cpp

```

1  /**
2   * Sample code for callback usage.
3   */
4  #include <cepton_sdk_api.hpp>
5
6  // Sample global callback.
7  void on_image_frame(cepton_sdk::SensorHandle handle, std::size_t n_points,
8                     const cepton_sdk::SensorImagePoint *c_image_points) {}
9
10 // Sample member callback.
11 class FramesListener {
12 public:
13     void on_image_frame(cepton_sdk::SensorHandle handle, std::size_t n_points,
14                        const cepton_sdk::SensorImagePoint *c_image_points) {}
15 };
16
17 int main(int argc, char **argv) {
18     // Initialize
19     cepton_sdk::api::check_error(cepton_sdk::api::initialize());
20     cepton_sdk::api::SensorImageFrameCallback callback;
21     cepton_sdk::api::check_error(callback.initialize());
22
23     // Listen lambda
24     callback.listen([](cepton_sdk::SensorHandle handle, std::size_t n_points,
25                      const cepton_sdk::SensorImagePoint *c_image_points) {});
26
27     // Listen global function
28     callback.listen(on_image_frame);
29
30     // Listen member function

```

(continues on next page)

(continued from previous page)

```

31 FramesListener frames_listener;
32 callback.listen(&frames_listener, &FramesListener::on_image_frame);
33 }

```

## 10.4 Error

Listing 4: samples/error.cpp

```

1  /**
2   * Sample code for error callback usage.
3   */
4  #include <cepton_sdk_api.hpp>
5
6  int main(int argc, char** argv) {
7      // Initialize
8      cepton_sdk::api::SensorErrorCallback error_callback;
9      error_callback.listen(
10         [&](cepton_sdk::SensorHandle handle,
11            const cepton_sdk::SensorError& error) { throw error; });
12     cepton_sdk::api::check_error(cepton_sdk::initialize(
13         CEPTON_SDK_VERSION, cepton_sdk::create_options(),
14         error_callback.global_on_callback, &error_callback));
15 }

```

## 10.5 Frame

Listing 5: samples/frame.cpp

```

1  /**
2   * Sample code for custom frame accumulation.
3   */
4  #include <cepton_sdk_api.hpp>
5
6  int main(int argc, char **argv) {
7      std::string capture_path;
8      if (argc >= 2) capture_path = argv[1];
9
10     // Initialize
11     auto options = cepton_sdk::create_options();
12     cepton_sdk::api::check_error(
13         cepton_sdk::api::initialize(options, capture_path));
14     cepton_sdk::api::SensorImageFrameCallback callback;
15     cepton_sdk::api::check_error(callback.initialize());
16
17     // Get sensor
18     while (cepton_sdk::get_n_sensors() == 0)
19         cepton_sdk::api::check_error(cepton_sdk::api::wait(0.1f));
20     cepton_sdk::SensorInformation sensor_info;
21     cepton_sdk::api::check_error(
22         cepton_sdk::get_sensor_information_by_index(0, sensor_info));
23 }

```

(continues on next page)



(continued from previous page)

```

24 // Create accumulator
25 auto frame_options = cepton_sdk::create_frame_options();
26 frame_options.mode = CEPTON_SDK_FRAME_TIMED;
27 frame_options.length = 0.1f;
28 cepton_sdk::util::FrameAccumulator accumulator(sensor_info);
29 cepton_sdk::api::check_error(accumulator.set_options(frame_options));
30 callback.listen(
31     [&](cepton_sdk::SensorHandle handle, std::size_t n_points,
32         const cepton_sdk::SensorImagePoint *const c_image_points) {
33         if (handle != sensor_info.handle) return;
34         accumulator.add_points(n_points, c_image_points);
35     });
36
37 // Listen
38 accumulator.callback.listen(
39     [&](int n_points,
40         const cepton_sdk::SensorImagePoint *const c_image_points) {
41         std::printf("Received %i points\n", n_points);
42     });
43
44 cepton_sdk::api::check_error(cepton_sdk::api::wait(5.0f));
45 }

```

## 10.6 Network

Listing 6: samples/network.cpp

```

1 /**
2  * Sample code for custom networking.
3  */
4 #include <asio.hpp>
5
6 #include <cepton_sdk_api.hpp>
7
8 using asio::ip::udp;
9
10 class SocketListener {
11 public:
12     SocketListener() : m_socket(m_io_service, udp::v4()) {
13         m_socket.set_option(asio::socket_base::reuse_address(true));
14         m_socket.bind(udp::endpoint(udp::v4(), 8808));
15     }
16
17     void run() {
18         listen();
19         m_io_service.run();
20     }
21
22     void listen() {
23         m_socket.async_receive_from(
24             asio::buffer(m_buffer), m_end_point,
25             [this](const asio::error_code& error, std::size_t buffer_size) {
26                 if (buffer_size == 0) return;
27                 if (error == asio::error::operation_aborted) return;

```

(continues on next page)

(continued from previous page)

```

28     const CeptonSensorHandle handle =
29         m_end_point.address().to_v4().to_ulong();
30     // For more accurate timestamps, a separate network receive thread
31     // should be
32     // used.
33     const int64_t timestamp = cepton_sdk::util::get_timestamp_usec();
34     cepton_sdk::api::check_error(cepton_sdk::mock_network_receive(
35         handle, timestamp, m_buffer.data(), buffer_size));
36     listen();
37     });
38 }
39
40 private:
41     asio::io_service m_io_service;
42     udp::socket m_socket;
43     udp::endpoint m_end_point;
44     std::array<uint8_t, 4096> m_buffer;
45 };
46
47 int main() {
48     // Initialize sdk
49     auto options = cepton_sdk::create_options();
50     options.control_flags |= CEPTON_SDK_CONTROL_DISABLE_NETWORK;
51     options.frame.mode = CEPTON_SDK_FRAME_COVER;
52     cepton_sdk::api::check_error(cepton_sdk::api::initialize(options));
53
54     // Listen for points
55     cepton_sdk::api::SensorImageFrameCallback callback;
56     cepton_sdk::api::check_error(callback.initialize());
57     callback.listen([](cepton_sdk::SensorHandle handle, std::size_t n_points,
58         const cepton_sdk::SensorImagePoint* c_image_points) {
59         std::printf("Received %i points from sensor %lli\n", (int)n_points,
60             (long long)handle);
61     });
62
63     SocketListener listener;
64     listener.run();
65 }

```

## 10.7 Replay

Listing 7: samples/replay.cpp

```

1  /**
2   * Sample code for custom packet replaying.
3   */
4  #include <cepton_sdk/capture.hpp>
5  #include <cepton_sdk_api.hpp>
6
7  class CaptureReplay {
8  public:
9      CaptureReplay(const std::string& path) {
10         cepton_sdk::api::check_error(m_capture.open_for_read(path));
11     }

```

(continues on next page)

(continued from previous page)

```

12     cepton_sdk::api::check_error(
13         cepton_sdk_set_control_flags(CEPTON_SDK_CONTROL_DISABLE_NETWORK,
14             CEPTON_SDK_CONTROL_DISABLE_NETWORK));
15     cepton_sdk::api::check_error(cepton_sdk_clear());
16 }
17
18 ~CaptureReplay() {
19     m_capture.close();
20     if (cepton_sdk_is_initialized()) {
21         cepton_sdk::api::check_error(cepton_sdk_clear());
22     }
23 }
24
25 void run() {
26     while (true) {
27         cepton_sdk::Capture::PacketHeader header;
28         const uint8_t* data;
29         cepton_sdk::api::check_error(m_capture.next_packet(header, data));
30
31         const cepton_sdk::SensorHandle handle =
32             (cepton_sdk::SensorHandle)header.ip_v4 |
33             CEPTON_SENSOR_HANDLE_FLAG MOCK;
34         cepton_sdk::api::check_error(cepton_sdk_mock_network_receive(
35             handle, header.timestamp, data, header.data_size));
36     }
37 }
38
39 private:
40     cepton_sdk::Capture m_capture;
41 };
42
43 int main(int argc, char** argv) {
44     if (argc < 2) return -1;
45     const std::string capture_path = argv[1];
46
47     // Initialize sdk
48     auto options = cepton_sdk::create_options();
49     options.control_flags |= CEPTON_SDK_CONTROL_DISABLE_NETWORK;
50     options.frame.mode = CEPTON_SDK_FRAME_COVER;
51     cepton_sdk::api::check_error(cepton_sdk::api::initialize(options));
52
53     // Listen for points
54     cepton_sdk::api::SensorImageFrameCallback callback;
55     cepton_sdk::api::check_error(callback.initialize());
56     callback.listen([](cepton_sdk::SensorHandle handle, std::size_t n_points,
57         const cepton_sdk::SensorImagePoint* c_image_points) {
58         std::printf("Received %i points from sensor %lli\n", (int)n_points,
59             (long long)handle);
60     });
61
62     CaptureReplay replay(capture_path);
63     replay.run();
64 }

```

## 10.8 Sensor

Listing 8: samples/sensor.cpp

```
1  /**
2   * Sample code for sensor information.
3   */
4  #include <string>
5
6  #include <cepton_sdk_api.hpp>
7
8  int main(int argc, char **argv) {
9      std::string capture_path;
10     if (argc >= 2) capture_path = argv[1];
11
12     // Initialize
13     auto options = cepton_sdk::create_options();
14     cepton_sdk::api::check_error(
15         cepton_sdk::api::initialize(options, capture_path));
16     cepton_sdk::api::check_error(cepton_sdk::api::wait(5.0f));
17
18     // Get all sensors
19     const int n_sensors = cepton_sdk::get_n_sensors();
20     for (int i = 0; i < n_sensors; ++i) {
21         cepton_sdk::SensorInformation sensor_info;
22         cepton_sdk::api::check_error(
23             cepton_sdk::get_sensor_information_by_index(i, sensor_info));
24         std::printf("%i: %s\n", (int)sensor_info.serial_number,
25                     sensor_info.model_name);
26     }
27 }
```

## 10.9 Stray

Listing 9: samples/stray.cpp

```
1  /**
2   * Sample code for stray filter usage
3   */
4  #include <vector>
5
6  #include <cepton_sdk_api.hpp>
7
8  int main(int argc, char **argv) {
9      std::string capture_path;
10     if (argc >= 2) capture_path = argv[1];
11
12     // Initialize
13     auto options = cepton_sdk::create_options();
14     cepton_sdk::api::check_error(
15         cepton_sdk::api::initialize(options, capture_path));
16
17     cepton_sdk::api::SensorImageFrameCallback callback;
18     cepton_sdk::api::check_error(callback.initialize());
```

(continues on next page)

(continued from previous page)

```

19 cepton_sdk::util::StrayFilter filter;
20 callback.listen([&](cepton_sdk::SensorHandle handle, std::size_t n_points,
21                    const cepton_sdk::SensorImagePoint *c_image_points) {
22     // Get sensor
23     cepton_sdk::SensorInformation sensor_info;
24     cepton_sdk::api::check_error(
25         cepton_sdk::get_sensor_information(handle, sensor_info));
26
27     // Copy points to buffer
28     std::vector<cepton_sdk::SensorImagePoint> image_points;
29     image_points.insert(image_points.begin(), c_image_points,
30                        c_image_points + n_points);
31
32     // Filter stray
33     filter.init(sensor_info);
34     filter.run(n_points, image_points.data());
35 });
36
37 // Run
38 cepton_sdk::api::check_error(cepton_sdk::api::wait(5.0f));
39 }

```

## 10.10 Transform

Listing 10: samples/transform.cpp

```

1  /**
2   * Sample code for transforming 3d points.
3   */
4  #include "cepton_sdk_api.hpp"
5
6  int main() {
7     // Create transform
8     std::array<float, 3> translation = {0.0f, 0.0f, 1.0f};
9     std::array<float, 4> rotation = {1.0f, 0.0f, 0.0f, 0.0f};
10    auto compiled_transform = cepton_sdk::util::CompiledTransform::create(
11        translation.data(), rotation.data());
12
13    // Apply transform
14    cepton_sdk::util::SensorPoint point = {};
15    compiled_transform.apply(point.x, point.y, point.z);
16    printf("Point: [%f, %f, %f]\n", point.x, point.y, point.z);
17 }

```



## C

cepton\_sdk::api::check\_error (C++ function), 19  
 cepton\_sdk::api::default\_on\_error (C++ function), 19  
 cepton\_sdk::api::get\_sensor\_information\_by\_serial\_number (C++ function), 20  
 cepton\_sdk::api::get\_sensor\_serial\_numbers (C++ function), 20  
 cepton\_sdk::api::get\_time (C++ function), 19  
 cepton\_sdk::api::has\_sensor\_by\_serial\_number (C++ function), 20  
 cepton\_sdk::api::initialize (C++ function), 19  
 cepton\_sdk::api::is\_end (C++ function), 19  
 cepton\_sdk::api::is\_live (C++ function), 19  
 cepton\_sdk::api::log\_error (C++ function), 19  
 cepton\_sdk::api::NetworkPacketCallback (C++ class), 20  
 cepton\_sdk::api::NetworkPacketCallback::~NetworkPacketCallback (C++ function), 20  
 cepton\_sdk::api::NetworkPacketCallback::~deinitialize (C++ function), 20  
 cepton\_sdk::api::NetworkPacketCallback::~initialize (C++ function), 20  
 cepton\_sdk::api::SensorErrorCallback (C++ class), 19  
 cepton\_sdk::api::SensorErrorCallback::global\_on\_callback (C++ function), 20  
 cepton\_sdk::api::SensorImageFrameCallback (C++ class), 20  
 cepton\_sdk::api::SensorImageFrameCallback::~SensorImageFrameCallback (C++ function), 20  
 cepton\_sdk::api::SensorImageFrameCallback::~deinitialize (C++ function), 20  
 cepton\_sdk::api::SensorImageFrameCallback::~initialize (C++ function), 20  
 cepton\_sdk::api::wait (C++ function), 19  
 cepton\_sdk::capture\_replay::close (C++ function), 17  
 cepton\_sdk::capture\_replay::get\_enable\_loop (C++ function), 17  
 cepton\_sdk::capture\_replay::get\_filename (C++ function), 17  
 cepton\_sdk::capture\_replay::get\_length (C++ function), 17  
 cepton\_sdk::capture\_replay::get\_position (C++ function), 17  
 cepton\_sdk::capture\_replay::get\_speed (C++ function), 17  
 cepton\_sdk::capture\_replay::get\_start\_time (C++ function), 17  
 cepton\_sdk::capture\_replay::is\_end (C++ function), 17  
 cepton\_sdk::capture\_replay::is\_open (C++ function), 17  
 cepton\_sdk::capture\_replay::is\_running (C++ function), 18  
 cepton\_sdk::capture\_replay::open (C++ function), 17  
 cepton\_sdk::capture\_replay::pause (C++ function), 18  
 cepton\_sdk::capture\_replay::resume (C++ function), 18  
 cepton\_sdk::capture\_replay::resume\_blocking (C++ function), 17  
 cepton\_sdk::capture\_replay::resume\_blocking\_once (C++ function), 17  
 cepton\_sdk::capture\_replay::seek (C++ function), 17  
 cepton\_sdk::capture\_replay::set\_enable\_loop (C++ function), 17  
 cepton\_sdk::capture\_replay::set\_speed (C++ function), 17  
 cepton\_sdk::clear (C++ function), 15  
 cepton\_sdk::Control (C++ type), 7  
 cepton\_sdk::create\_frame\_options (C++ function), 8  
 cepton\_sdk::create\_options (C++ function), 9  
 cepton\_sdk::deinitialize (C++ function), 9  
 cepton\_sdk::FpNetworkReceiveCallback (C++ type), 15  
 cepton\_sdk::FpSensorErrorCallback (C++

*type*), 9  
 cepton\_sdk::FrameMode (C++ *type*), 8  
 cepton\_sdk::FrameOptions (C++ *type*), 8  
 cepton\_sdk::get\_control\_flags (C++ *function*), 9  
 cepton\_sdk::get\_error (C++ *function*), 4  
 cepton\_sdk::get\_error\_code\_name (C++ *function*), 4  
 cepton\_sdk::get\_frame\_length (C++ *function*), 9  
 cepton\_sdk::get\_frame\_mode (C++ *function*), 9  
 cepton\_sdk::get\_n\_sensors (C++ *function*), 12  
 cepton\_sdk::get\_port (C++ *function*), 9  
 cepton\_sdk::get\_sensor\_handle\_by\_serial\_number (C++ *function*), 12  
 cepton\_sdk::get\_sensor\_information (C++ *function*), 12  
 cepton\_sdk::get\_sensor\_information\_by\_index (C++ *function*), 12  
 cepton\_sdk::has\_control\_flag (C++ *function*), 9  
 cepton\_sdk::initialize (C++ *function*), 9  
 cepton\_sdk::is\_error\_code (C++ *function*), 4  
 cepton\_sdk::is\_fault\_code (C++ *function*), 4  
 cepton\_sdk::listen\_image\_frames (C++ *function*), 13  
 cepton\_sdk::listen\_network\_packets (C++ *function*), 15  
 cepton\_sdk::Options (C++ *type*), 9  
 cepton\_sdk::SensorError (C++ *class*), 3  
 cepton\_sdk::SensorError::code (C++ *member*), 3  
 cepton\_sdk::SensorError::is\_error (C++ *function*), 3  
 cepton\_sdk::SensorError::is\_fault (C++ *function*), 3  
 cepton\_sdk::SensorError::msg (C++ *member*), 3  
 cepton\_sdk::SensorError::name (C++ *function*), 3  
 cepton\_sdk::SensorError::operator bool (C++ *function*), 3  
 cepton\_sdk::SensorError::operator SensorErrorCode (C++ *function*), 3  
 cepton\_sdk::SensorError::SensorError (C++ *function*), 3  
 cepton\_sdk::SensorErrorCode (C++ *type*), 4  
 cepton\_sdk::SensorHandle (C++ *type*), 11  
 cepton\_sdk::SensorImagePoint (C++ *type*), 13  
 cepton\_sdk::SensorInformation (C++ *type*), 11  
 cepton\_sdk::SensorModel (C++ *type*), 11  
 cepton\_sdk::set\_control\_flags (C++ *function*), 9  
 cepton\_sdk::set\_frame\_options (C++ *function*), 9  
 cepton\_sdk::set\_port (C++ *function*), 9  
 cepton\_sdk::unlisten\_image\_frames (C++ *function*), 14  
 cepton\_sdk::unlisten\_network\_packets (C++ *function*), 15  
 cepton\_sdk::util::Callback (C++ *class*), 22  
 cepton\_sdk::util::Callback::clear (C++ *function*), 22  
 cepton\_sdk::util::Callback::emit (C++ *function*), 22  
 cepton\_sdk::util::Callback::global\_on\_callback  
 cepton\_sdk::util::Callback::listen (C++ *function*), 22  
 cepton\_sdk::util::Callback::operator ()  
 cepton\_sdk::util::CompiledTransform (C++ *class*), 21  
 cepton\_sdk::util::CompiledTransform::apply (C++ *function*), 22  
 cepton\_sdk::util::CompiledTransform::create (C++ *function*), 22  
 cepton\_sdk::util::convert\_image\_point\_to\_point (C++ *function*), 21  
 cepton\_sdk::util::convert\_sensor\_image\_point\_to\_point (C++ *function*), 21  
 cepton\_sdk::util::FrameAccumulator (C++ *class*), 23  
 cepton\_sdk::util::FrameAccumulator::add\_points (C++ *function*), 23  
 cepton\_sdk::util::FrameAccumulator::callback (C++ *member*), 23  
 cepton\_sdk::util::FrameAccumulator::clear (C++ *function*), 23  
 cepton\_sdk::util::FrameAccumulator::FrameAccumulator (C++ *function*), 23  
 cepton\_sdk::util::FrameAccumulator::get\_options (C++ *function*), 23  
 cepton\_sdk::util::FrameAccumulator::set\_options (C++ *function*), 23  
 cepton\_sdk::util::FrameDetector (C++ *class*), 23  
 cepton\_sdk::util::FrameDetector::add\_point (C++ *function*), 23  
 cepton\_sdk::util::FrameDetector::frame\_found (C++ *member*), 23  
 cepton\_sdk::util::FrameDetector::frame\_idx (C++ *member*), 23  
 cepton\_sdk::util::FrameDetector::frame\_x (C++ *member*), 23  
 cepton\_sdk::util::FrameDetector::FrameDetector (C++ *function*), 23



cepton\_sdk::util::FrameDetector::get\_options  
(C++ *function*), [23](#)

cepton\_sdk::util::FrameDetector::reset  
(C++ *function*), [23](#)

cepton\_sdk::util::FrameDetector::set\_options  
(C++ *function*), [23](#)

cepton\_sdk::util::get\_timestamp\_usec  
(C++ *function*), [21](#)

cepton\_sdk::util::SensorPoint (C++ *class*),  
[21](#)

cepton\_sdk::util::SensorPoint::x (C++  
*member*), [21](#)

cepton\_sdk::util::SensorPoint::y (C++  
*member*), [21](#)

cepton\_sdk::util::SensorPoint::z (C++  
*member*), [21](#)